Flask-Diced Documentation

Release 0.2

Philip Xu

March 30, 2016

Contents

Flask-Diced is a set of helper classes for Flask that generates CRUD views and registers them to blueprint/application.

Flask-Diced provides:

- **D**etail view
- **I**ndex view
- **C**reate view
- **E**dit view
- **D**elete view

CHAPTER 1

Flask-Diced Might Not Be What You Want

Flask-Diced is opinionated, it assumes:

- the model object has a `save` method for data persistence and `delete` method to remove itself.
- the model can be created with no required arguments.
- the form used for creation and editing are in the style of Flask-WTF's, specifically, Flask-Diced expects the form accepts a named parameter `obj` to pass in the initial value for editing as well as `validate_on_submit` method and `populate_obj` method on the form class. In short, just use *Flask-WTF*.
- the client should be redirected when done for POST requests.
- views should have minimal business logic.

Flask-Diced is Extensible

Flask-Diced is designed in a way that customizations can be easily done. All properties and methods can be overridden for customization.

Flask-Diced can be customized to the point that the assumptions described in last section are refer to the default implementation and will no longer hold true if you customize relevant parts of it.

e.g.,

Want to change how the objects list is fetched?

Override `query_all`

Want to change the name of endpoint of the edit view?

Redefine `edit_endpoint`

Want to use your own view function or control how views are registered?

Override respective view/register methods.

CHAPTER 3

Installation

Flask-Diced is on PyPI.

```
pip install Flask-Diced
```

CHAPTER 4

License

BSD New, see LICENSE for details.

CHAPTER 5

Links

- Documentation
- Issue Tracker
- Source Package @ PyPI
- Mercurial Repository @ bitbucket
- Git Repository @ Github
- Git Repository @ Gitlab
- Development Version

CHAPTER 6

Example

The python code of an example application is included entirely here.

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_wtf import Form
from wtforms import TextField, SubmitField
from wtforms.fields.html5 import EmailField
from wtforms.validators import DataRequired, Email, ValidationError

from flask_diced import Diced, persistence_methods


app = Flask(__name__)

# Need this for WTForm CSRF protection
app.config['SECRET_KEY'] = 'no one knows'

# Need this for SQLAlchemy
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///:memory:'

db = SQLAlchemy(app)


# persistence_methods is a class decorator that adds save and delete methods
@persistence_methods(db)
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True)
    email = db.Column(db.String(120), unique=True)


def unique(model, column, message='already exists'):
    def unique_validator(form, field):
        obj = model.query.filter(column == field.data).first()
        if obj and obj.id != form._obj_id:
            raise ValidationError(message)
    return unique_validator


class UserForm(Form):
    username = TextField('Username',
                          [DataRequired(), unique(User, User.username)])
    email = EmailField('Email',
                       [DataRequired(), Email(), unique(User, User.email)])

    def __init__(self, **kwargs):
        super(UserForm, self).__init__(**kwargs)
        self._obj_id = kwargs['obj'].id if 'obj' in kwargs else None


class CreateUserForm(UserForm):
    submit = SubmitField('Create')


class EditUserForm(UserForm):
    submit = SubmitField('Update')


class DeleteForm(Form):
    submit = SubmitField('Delete')


# view decorator that does nothing, for showing how to add decorators to views
def no_op_decorator(view):
    return view


# create a view generator for User model, these two arguments for decorators
# are here for demonstration purpose and are not mandatory
user_view = Diced(
    model=User,
    create_form_class=CreateUserForm,
    edit_form_class=EditUserForm,
    delete_form_class=DeleteForm,
    index_decorators=[no_op_decorator],
    edit_decorators=[no_op_decorator, no_op_decorator],
)


# Register on application directly, works on Blueprint as well
user_view.register(app)


if __name__ == '__main__':
    db.create_all()
    app.run(debug=True)
```

In this example, `Diced` is used directly and required attributes are passed in when creating an instance of the class, another way is to subclass and define required attributes as class attributes, as in:

```
class UserView(Diced):
    model = User
    create_form_class = CreateUserForm
    edit_form_class = EditUserForm
    delete_form_class = DeleteForm

user_view = UserView()
user_view.register(app)
```

CHAPTER 7

API

Flask-Diced - CRUD views generator for Flask

class `flask_diced.`**`Detail`**

Bases: `object`

detail view mixin

`detail_decorators` = ()

decorators to be applied to detail view

`detail_endpoint` = **'detail'**

the endpoint for the detail view URL rule

`detail_rule` = **'/<int:pk>/'**

the URL rule for the detail view

`detail_template`

default template name for detail view

generated with *`object_name`* and *`detail_endpoint`*

`detail_view`(*pk*)

detail view function

Parameters **`pk`** – the primary key of the model to be shown.

`register_detail_view`(*blueprint*)

register detail view to blueprint

Parameters **`blueprint`** – the Flask Blueprint or Application object to which the detail view will be registered.

class `flask_diced.`**`Index`**

Bases: `object`

index view mixin

`index_decorators` = ()

decorators to be applied to index view

`index_endpoint` = **'index'**

the endpoint for the index view URL rule

`index_rule` = **'/'**

the URL rule for the index view

`index_template`

default template name for index view

generated with *`object_name`* and *`index_endpoint`*

`index_view``()`
index view function

`register_index_view``(`*blueprint*`)`
register index view to blueprint

Parameters **`blueprint`** – the Flask Blueprint or Application object to which the index view will be registered.

class `flask_diced.`**`Create`**
Bases: `object`

create view mixin

`create_decorators` **= ()**
decorators to be applied to create view

`create_endpoint` **= 'create'**
the endpoint for the create view URL rule

`create_flash_message` **= None**
the message to be flashed for the next request when done

`create_form_class` **= None**
the form class for new object, with Flask-WFT compatible API, this attribute is **mandatory** if create view is enabled unless the default view *`create_view()`* is overridden and does not use it

`create_form_name` **= 'form'**
the name for variable representing the form in template

`create_redirect_to_view` **= '.index'**
the name of view to redirect the client to when done

`create_redirect_url`
the url the client will be redirected to when done

the default value is the url of *`create_redirect_to_view`*

`create_rule` **= '/create/'**
the URL rule for the create view

`create_template`
default template name for create view

generated with *`object_name`* and *`create_endpoint`*

`create_view``()`
create view function

`register_create_view``(`*blueprint*`)`
register create view to blueprint

Parameters **`blueprint`** – the Flask Blueprint or Application object to which the create view will be registered.

class `flask_diced.`**`Edit`**
Bases: `object`

edit view mixin

`edit_decorators` **= ()**
decorators to be applied to edit view

`edit_endpoint` = **'edit'**
: the endpoint for the edit view URL rule

`edit_flash_message` = **None**
: the message to be flashed for the next request when done

`edit_form_class` = **None**
: the form class for editing object, with Flask-WFT compatible API, this attribute is **mandatory** if edit view is enabled unless the default view *`edit_view()`* is overridden and does not use it

`edit_form_name` = **'form'**
: the name for variable representing the form in template.

`edit_redirect_to_view` = **'.index'**
: the name of view to redirect the client to when done

`edit_redirect_url`
: the url the client will be redirected to when done

  the default value is the url of *`edit_redirect_to_view`*

`edit_rule` = **'/<int:pk>/edit/'**
: the URL rule for the edit view

`edit_template`
: default template name for edit view

  generated with *`object_name`* and *`edit_endpoint`*

`edit_view`(*pk*)
: edit view function

  Parameters **`pk`** – the primary key of the model to be edited.

`register_edit_view`(*blueprint*)
: register edit view to blueprint

  Parameters **`blueprint`** – the Flask Blueprint or Application object to which the edit view will be registered.

class `flask_diced.`**`Delete`**
: Bases: `object`

  delete view mixin

`delete_decorators` = **()**
: decorators to be applied to delete view

`delete_endpoint` = **'delete'**
: the endpoint for the delete view URL rule

`delete_flash_message` = **None**
: the message to be flashed for the next request when done

`delete_form_class` = **None**
: the form class for deletion confirmation, should validate if confirmed this attribute is **mandatory** if delete view is enabled unless the default view *`delete_view()`* is overridden and does not use it

`delete_form_name` = **'form'**
: the name for variable representing the form in template

`delete_redirect_to_view` = **'.index'**
: the name of view to redirect the client to when done

`delete_redirect_url`
the url the client will be redirected to when done

the default value is the url of *`delete_redirect_to_view`*

`delete_rule` **= '/<int:pk>/delete/'**
the URL rule for the delete view

`delete_template`
default template name for delete view

generated with *`object_name`* and *`delete_endpoint`*

`delete_view`(*pk*)
delete view function

Parameters **`pk`** – the primary key of the model to be deleted.

`register_delete_view`(*blueprint*)
register delete view to blueprint

Parameters **`blueprint`** – the Flask Blueprint or Application object to which the delete view will be registered.

class `flask_diced.`**`Base`**(***options*)
Bases: `object`

base class with properties and methods used by mixins

`__init__`(***options*)
create an instance of view generator

all keyword arguments passed in will be set as the instance's attribute if the name is not starting with '_'

`exclude_views` **= set([])**
views that will not be registered when *`register()`* is called, even if they are also listed in *`views`*

`model` **= None**
the model class, this attribute is **mandatory**

`object_list_name`
default name for variable representing list of objects in templates

generated with *`object_name`* in detault implementation.

`object_name`
default name for variable representing object in templates

generated with the name of model class in detault implementation.

`query_all`()
returns all objects

`query_object`(*pk*)
returns the object with matching `pk`

`register`(*blueprint*)
register all enabled views to the `blueprint`

Parameters **`blueprint`** – the Flask Blueprint or Application object to which enalbed views will be registered.

`views` **= set(['edit', 'index', 'create', 'detail', 'delete'])**
views that will be registered when *`register()`* is called

class `flask_diced.`**`Diced`**`(`*`**options`*`)`

Bases: *`flask_diced.Detail`*, *`flask_diced.Index`*, *`flask_diced.Create`*, *`flask_diced.Edit`*, *`flask_diced.Delete`*, *`flask_diced.Base`*

CRUD views generator

`flask_diced.`**`persistence_methods`**`(`*`datastore`*`)`

class decorator that adds persistence methods to the model class

Parameters **`datastore`** – SQLAlchemy style datastore, should sopport `datastore.session.add()`, `datastore.session.delete()` and `datastore.session.commit()` for model persistence.

Two persistence methods will be added to the decorated class

`save(self, commit=True)` the save method

`delete(self, commit=True)` the delete method

CHAPTER 8

Changelog

8.1 Version 0.2

- Removed test runner dependency in end user installation

8.2 Version 0.1

- Initial public release

Python Module Index